NetarchiveSuite Installation Manual

Printer friendly version

Contents

Introduction

edit

This manual describes how to install and configure the NetarchiveSuite web archive software package. It includes description of how to obtain and install required libraries, how to install the software on separate machines, what command line options and configuration file changes are necessary, and how to start the programs. It then goes on to explain the other parameters available for tuning the behaviour of NetarchiveSuite. It does not explain how to extend the functionality of the system (see the Developer Manual for this) or how to use the running system (see the User Manual for this).

The intended audience of this manual is system administrators who will be responsible for the actual installation and setup of NetarchiveSuite as well as technical personnel responsible for proper operation of NetarchiveSuite. Knowledge of Unix system administration is expected, and some familiarity with XML and Java is an advantage.

Configuration basics

edit

Almost all configuration of NetarchiveSuite applications is done by changing the `settings.xml` file, which by default is located in the `conf` directory. This file is organized in a hierarchy with settings for the various modules and applications having their own parts. All settings can also be set from the command line by specifying the -D option with a dotted path of the XML path. For instance, using the command line option `-Dsettings.common.tempDir=/tmp` corresponds to changing the XML file to read (in part)

```
<settings>
    <common>
        <!-- Common temporary directory for all applications. -->
        <tempDir>/tmp</tempDir>
...
```

Using command line options override whatever is set in the `settings.xml` file, but allows only one value per setting. In the `settings.xml` file, you can specify several values by repeating a section. For instance,

Whether to use a `settings.xml` file per application or per machine or just have the same file for all machines is up to you. In this manual, we will assume a shared `settings.xml` file for all machines and define the settings that vary using command-line options. Depending on how you choose to perform your installation, you may want to have a different approach.

Choose an installation scenario

edit

NetarchiveSuite can be installed in a number of different ways, with varying numbers of machines on different sites. There is a number of separate applications in play, most of which can be put on separate machines as needed. To keep clear what is necessary for which setups, we will consider the following types of setup:

- A. **Single-machine setup**. This corresponds to the setup used in the QuickStart Manual, where all applications run on the same machine, and file transfer can be done simple by copying files locally. It is the simplest setup, but does not scale very well. Note that the scripts used in QuickStart Manual resets the system at every restart, including deleting all harvested material. Obviously, this is not the intent for a running installation, so those scripts cannot be used in production environments as they are.
- B. **Single-site setup**. In this scenario, multiple machines are involved, necessitating file transfer between machines and multiple installations of the code. However, the machines are expected to be within the same firewall, so port setup should be no problem.
- C. **Single-site setup with duplicate archive**. This expands on the single-site setup in that more than one copy of the archives are used, using the concept of separate "Locations" to indicate the duplicates.
- D. **Multi-site setup**. When more than one site is involved, separated by firewalls, extra issues of opening ports and specifying the correct site come into play. This is the most complex scenario, but also the more secure against systematic errors, hacking, and other disasters.

Setups C and D involves having a distributed bitarchive. In these setups we have the the bitarchive distributed on at least two Locations, here called LocationA and LocationB. One of them must be designated as the Location that by default executes batch jobs -- typically the Location with the greater amount of processing power. These Location informations must be written to the general settings.xml before deployment:

```
<arcrepository>
  ...
  <!-- The names of all bit archive locations in the
       environment, e.g., "LocationA" and "LocationB". -->
  <location>
    <name>LocationA</name>
  </location>
  <location>
    <name>LocationB</name>
  </location>
  <!-- Default bit archive to use for batch jobs (if none is specified)
-->
  <batchLocation>LocationA</batchLocation>
</arcrepository>
```

Choose a JMS broker

edit

The NetarchiveSuite requires the use of a JMS broker. The installation and startup of a JMSbroker is described in Appendix A. In the below extract of conf/settings.xml, the JMSbroker resides at machine1.domain, and listens for messages on port 7676. You must also select a JMS environmentName. This allows you have more than one running installation of the NetarchiveSuite, each with its own environmentName, and makes it easy to cleanup the JMS queues associated with a given environmentName. The NetarchiveSuite currently only supports one kind of JMS broker, so only the 'broker','port', and 'environmentName' can be changed.

```
<jms>
    <!-- Selects the broker vendor to be used. -->
            <class>SunMQ</class>
            <!-- The JMS broker host contacted by the JMS connection
-->
            <broker>localhost</broker>
            <!-- The port the JMS connection should use -->
            <port>7676</port>
            <!-- The name of the environment in which this code is
running, e.g.
                 PROD, RELEASETEST, NHC,... Common prefix to all JMS
channels
                  -->
            <environmentName>PROD</environmentName>
        </jms>
```

Firewall note: The machine that runs the JMS broker must be accessible from all machines in the installation on not only port 7676, but also port 33700 (from RMI).

Choose the set of machines taking part in the

installation/deployment

edit

When you have chosen your setup, you must decide on the number of machines, you want to use in the deployment of the NetarchiveSuite. For setup A, the answer is of course one. For the setup B-D, the answer is more complicated.

At the Danish installation, we operate with 4 kinds of machines:

- Admin machine (one server): Here we deploy one or more BitarchiveMonitorApplications (one for each Location),

  one ArcrepositoryApplication, and one HarvestDefinitionApplication (running the GUI, and scheduler). The latter application is the only application using a database.
- harvester machines (one or more): Here we deploy the HarvesterControllerApplications and their associated SideKicks.
- bitarchive machines (one or more): These machines only run one BitarchiveApplication each.
- access servers (one or more): On these machines, we have the ViewerproxyApplication enabling us to browse in already stored webpages,

  and the IndexServerApplication. The latter must only be installed on one of the access-servers, as there can only be one in the system,

Apart from the HarvestControllerApplications and their associated SideKicks, there is no requirement that the applications are placed like this, but we will use it as an example throughout the rest of the manual. In the standard setup used in our test-environment, we have 9 machines:

```
1 bitarchive server (on Location A)
2 bitarchive servers (on Location B)

1 admin machine (placed on Location A)
2 harvester-machines (placed on Location A)
2 harvester-machines (placed on Location B)

1 access server (placed on Location A)
```

Configure monitoring (allocating JMX and RMI ports)

Monitoring the deployed NetarchiveSuite relies on JMX (Java Management Extensions). Each application in the NetarchiveSuite needs its own JMX-port and associated RMI-port, so they can be monitored from the NetarchiveSuite GUI, and using jconsole (see below). You need to select a range for the JMX-ports. In the example below, the chosen JMX/RMI-range begins at 8100.

Note: The RMI-ports for a certain JMX-port are assumed to be JMX-port-number + 100

Firewall Note: This requires that the admin-machine has access to each machine taking part in the deployment on ports 8100-8300.

You need to select a password for the JMX monitorRole, and replace the string "JMX_MONITOR_ROLE_PASSWORD_PLACEHOLDER" with the selected password in two files: the conf/jmxremote.password, and the settings file used. When starting the application we define the path to the jmx passwordfile on the commandline:

```
-Dsettings.common.jmx.passwordFile=INSTALLATION_DIR/conf/jmxremote.password
```

The JMX-ports are registered in the settings.xml used by the HarvestDefinitionApplication (GUI/Scheduler) in the deploy section of the settings.xml file:

```
<deploy>
  <jmxMonitorRolePassword>SELECTED_PASSWORD</jmxMonitorRolePassword>
  <numberOfHosts>NUMBER_OF_MACHINES_INVOLVED</numberOfHosts>
  <host1>
    <name>MACHINE_1</name>
    <jmxport>8100</jmxport>
    <jmxport>8101</jmxport>
  </host1>
  ...
  <hostX>
    <name>MACHINE_X</name>
    <jmxport>8100</jmxport>
    <jmxport>8101</jmxport>
  </hostX>
</deploy>
```

Additional configurations

edit

Configure notifications

NetarchiveSuite can send notifications of serious system warnings or failures to the system-owner by email. This is implemented using the Notifications plugin, see Appendix C. Several settings in the settings.xml must be changed for this to work:

The setting **settings.common.notifications.receiver** (recipient of notifications), **settings.common.notifications.sender** (the official sender of the email, and receiver of any bounces), and **settings.common.mail.server** (the proper mail-server to use):

```
<common>
  ...
  <notifications>
     <!-- Which class to instantiate to handle error notifications -->
     <class>dk.netarkivet.common.utils.EMailNotifications</class>
     <!-- The receiver of emails -->
     <receiver>example@netarkivet.dk</receiver>
     <!-- The stated sender of emails  (and receiver of bounces)-->
     <sender>example@netarkivet.dk</sender>
 </notifications>
 <!-- Settings for sending email. Currently mail is only used for email
notifications. -->
 <mail>
   <!-- The email server to use -->
   <server>examplesmtpserver.netarkivet.dk</server>
 </mail>
```

Select a file datatransfer method

You can currently choose between FTP or HTTP as the filetransfer method. The HTTP transfer method uses only a single copy per transfer, while the FTP method first copies the file to an FTP server and then copies it from there to the receiving side. Additionally, the HTTP transfer method reverts to simple filesystem copying whenever possible to optimize transfer speeds. However, to use HTTP transfers you must have ports open into most machines, which some may consider a security risk.

The FTP method requires one or more FTP-servers installed. (See Appendix A for further details). The XML below is a extract of a settings.xml, in which you have to replace serverName, userName, userPassword with proper values. This must be set for all applications.

```
<common>
  ...
  <remoteFile xsi:type="ftpremotefile">
    <!-- The class to use for RemoteFile objects. -->
    <class>dk.netarkivet.common.distribute.FTPRemoteFile</class>
    <!-- The default FTP-server used -->
    <serverName>hostname</serverName>
    <!-- The default FTP-server port used -->
    <serverPort>21</serverPort>
    <!-- The default FTP username -->
    <userName>exampleusername</userName>
    <!-- The default FTP password -->
    <userPassword>examplepassword</userPassword>
    <!-- The number of times FTPRemoteFile should try before giving up
         a copyTo operation. We augment FTP with checksum checks. -->
    <retries>3</retries>
  </remoteFile>
```

It is possible to use more than one FTP server, but each application can only use one. The FTP server that is used for a particular transfer is determined by the application that is sending a file. If you want to use more than one FTP-server, you must use different settings-files, or define the serverName and possibly also the userName and userPassword when starting the applications on the commandline with

```
  -Dsettings.commom.distribute.RemoteFile.serverName=FTP-server1
   -Dsettings.commom.distribute.RemoteFile.userName=ftpUser
   -Dsettings.commom.distribute.RemoteFile.userPassword=ftpPassword
```

Using HTTP as filetransfer method, you need to reserve a HTTP port on each machine per application. You can set this port on an application level on the commandline:
`-Dsettings.common.remoteFile.port=5442`

The following XML shows the the corresponding syntax in the `settings.xml` file:

```
<common>
  <remoteFile xsi:type="httpremotefile">
   <!-- The class to use for RemoteFile objects. -->
   <class>dk.netarkivet.common.distribute.HTTPRemoteFile</class>
   <!-- Port for embedded HTTP server -->
   <port>5442</port>
 </remoteFile>
```

Configure job-generation

The scheduling takes place every one minute, unless the previous scheduling is not finished yet. The scheduling interval cannot be changed. Scheduling amounts to searching for active harvestdefinitions, that is ready to have jobs generated, and submitted for harvesting. The job-generation procedure are governed by a set of settings prefixed by *settings.harvester.scheduler.*. These settings rule how large your crawljobs are going to be, and how long time they will take to complete. Note that harvestdefinitions consist of at least one DomainConfiguration, (containing a Heritrix setup, and a seed-list), and that there are two kinds: Snapshot Harvestdefinitions, and Selective Harvestdefinitions.

During scheduling, each harvest is split into a number of *crawl jobs*. This is done to keep Heritrix from using too much memory and to avoid that particularly slow or large domains cause harvests to take longer than necessary. In the job splitting part of the scheduling, the scheduler partitions a large number of DomainConfigurations into several crawljobs. Each crawljob can have only one Heritrix setup, so DomainConfigurations with different Heritrix setups will be split into different crawljobs. Additionally, a number of paramaters influence what configurations are put into which jobs, attempting to create jobs that cover a reasonable amount of domains of similar sizes.

If you don't want to have the harvests split into multiple jobs, you just need to set each of `jobs.maxRelativeSizeDifference`, `jobs.minAbsoluteSizeDifference`, `jobs.maxTotalSize`, and `configChunkSize` to a large number, such as MAX_LONG. Initially, we suggest you don't change these parameters, as the way the work together is subtle.

errorFactorPrevResult: Used when calculating expected size of a harvest of some domain during the job-creation process for snapshot harvests. This defines the factor by which we maximally expect domains that have previously been harvested to increase in size. The default value is 10.

errorFactorBestGuess: Used when calculating expected size of a harvest of some domain during job-creation process for a snapshot Harvests. This defines the factor by which we maximally expect domains that have previously been incompletely harvested or not harvested at all to increase in size. The default value is 20

expectedAverageBytesPerObject: How many bytes the average object is expected to be on domains where we don't know any better. This number should grow over time, as of end of 2005 empirical data shows 38000. Default is 38000.

maxDomainSize: Initial guess of #objects in an unknown domain. Default value is 5000

jobs.maxRelativeSizeDifference: The maximum allowed relative difference in expected number of objects retrieved in a single job definition. Set to MAX_LONG for no splitting.

jobs.minAbsoluteSizeDifference: Size differences for jobs below this threshold are ignored, regardless of the limits for the relative size difference. Set to MAX_LONG for no splitting. Default value is 2000.

jobs.maxTotalSize: When this limit is exceeded no more configurations may be added to a job. Set to MAX_LONG for no splitting. Default value is 2000000

configChunkSize: How many domain configurations we will process in one go before making jobs out of them. This amount of domains will be stored in memory at the same time. Set to MAX_LONG for no job splitting. The default value is 10000.

MAX_LONG refers to the number 2^63-1 or 9223372036854775807.

The actual deployment of the NetarchiveSuite

edit

This section describes one possible way to distribute the NetarchiveSuite software to a number of machines, but by no means the only one. In the examples below, we assume that `$NetarchiveSuiteDir` is set to the directory in which the NetarchiveSuite code is to be installed.

We assume that all machines in the chosen setup are unix servers. The procedure below may not work on other platforms. After creating the new settings to be used in the deployment of the software, zip together the NetarchiveSuite files including the new settings and copy the modified NetarchiveSuite.zip to all machines taking part in the deployment:

```
export USER=test
export MACHINES="machine1.domain1, machine2.domain1, ..
machine1.domain2, machine2.domain2"
for MACHINE in $MACHINES; do
  scp NetarchiveSuite.zip $USER@$MACHINE:$NetarchiveSuiteDir
  ssh $USER@$MACHINE "cd $NetarchiveSuiteDir && unzip
NetarchiveSuite.zip"
done
```

Standard commandline settings

The CLASSPATH

The CLASSPATH needed to start and run the java applications in NetarchiveSuite consists of 4 jarfiles, dk.netarkivet.harvester.jar, dk.netarkivet.archive.jar, dk.netarkivet.viewerproxy.jar, and dk.netarkivet.monitor.jar. The dk.netarkivet.common.jar and all our 3rd party dependencies need not be added explicitly to the CLASSPATH, as they are referenced indirectly in the jar-files.

```
export NetarchiveSuiteDir=/path/to/netarchiveSuite
export
CLASSPATH=$CLASSPATH:$NetarchiveSuiteDir/lib/dk.netarkivet.harvester.jar
export
CLASSPATH=$CLASSPATH:$NetarchiveSuiteDir/lib/dk.netarkivet.archive.jar
export
CLASSPATH=$CLASSPATH:$NetarchiveSuiteDir/lib/dk.netarkivet.viewerproxy.jar
export
CLASSPATH=$CLASSPATH:$NetarchiveSuiteDir/lib/dk.netarkivet.monitor.jar
```

Logging

We use the apache.commons.logging.framework, so we both need to point to the wanted logger-class (org.apache.commons.logging.impl.Jdk14Logger), and the logging configuration file. You may want to use different properties for different applications, especially when more that one application logs to the same logging directory. E.g. you want the change line `java.util.logging.FileHandler.pattern=./log/APPID%u.log` in the `conf/log.prop` file to something different.

```
export LOG_SETTINGS="-Dorg.apache.commons.logging.Log=\
  org.apache.commons.logging.impl.Jdk14Logger \
 -Djava.util.logging.config.file=$NetarchiveSuiteDir/conf/log.prop"
```

JMX settings

Each application has its own JMX- and RMI port. Here the JMX port is 8100 and the associated RMI port 8100 + 100. The monitoring facility in the GUI assumes that every RMI-port is JMXPort + 100. JMX also uses a password-file, which is the same throughout the installation ($NetarchiveSuiteDir/conf/jmxremote.password)

```
export JMX_SETTINGS=-Dsettings.common.jmx.port=8100 \
 -Dsettings.common.jmx.rmiPort=8200
```

Select the appropriate settings.file for the application

The conf/settings.xml (the new one configured to your environment) is probably OK for most applications. But you may need to use special purpose settings-files for some applications, e.g. BitarchiveApplications (since you can't allocate one than one fileDir on the commandline). The settings file used in an application can be specified by:

```
export
SETTING=-Ddk.netarkivet.settings.file=$NetarchiveSuiteDir/conf/settings.xml
```

JVM options

We need to set the maximum Java heap size to 1.5 Gbytes. You may use this to change that or add other JVM options.

```
export JAVA_OPTS=-Xmx1536m
```

Admin machine

On the admin machine, we have to start the following 4 applications:

- 1 HarvestdefinitionApplication (Starts the GUI and the scheduler).
- 2 instances of BitarchiveMonitorApplication (Controlling the access to a bitarchive at a single location), one for each location (e.g. LocationA, and LocationB).

- 1 ARCRepositoryApplication (this application handles access to the bitarchives).

Starting the HarvestDefinitionApplication

If you are not using the Embedded Derby Database, you can skip the following. This unzips the compressed fullhddb.jar located in $NetarchiveSuiteDir/harvestdefinitionbasedir/:

```
cd $NetarchiveSuiteDir/harvestdefinitionbasedir
unzip fullhddb.jar
```

Now we are ready to start the application:

```
cd $NetarchiveSuiteDir
export
APP=dk.netarkivet.harvester.webinterface.HarvestDefinitionApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP
```

Starting the BitarchiveMonitorApplication instances

In the general setup with a bitarchive distributed at two Locations, we have a BitarchiveMonitorApplication associated with each Location. Here the locations are "LocationA", and "LocationB". To distinguish the two instances from each other, we use the **settings.common.http.port** setting, which is used as a identifier (here we use 8081, 8082) as the two identifiers. there is also a credentials code in plain text associated with the bitarchive at each Location.

Start the monitor for bitarchive at 'LocationA' using "8081" as identifier thus:

```
cd $NetarchiveSuiteDir
export
APP_OPTIONS="-Dsettings.common.archive.bitarchive.thisLocation=LocationA
\
 -Dsettings.common.archive.bitarchive.thisCredentials=\
  SELECTED_CREDENTIALS_CODE_FOR_LOCATION_A \
 -Dsettings.common.http.port=8081"
export
APP=dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP_OPTIONS $APP
```

Start the monitor for archive at 'LocationB' using "8082" as identifier thus:

```
cd $NetarchiveSuiteDir
export
APP_OPTIONS="-Dsettings.common.archive.bitarchive.thisLocation=LocationB
\
 -Dsettings.common.archive.bitarchive.thisCredentials=\
  SELECTED_CREDENTIALS_CODE_FOR_LOCATION_B \
 -Dsettings.common.http.port=8082"
export
APP=dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP_OPTIONS $APP
```

- one ARCRepository (this takes care of all access to the bitarchives).

```
 cd $NetarchiveSuiteDir
 export
APP=dk.netarkivet.archive.arcrepository.ArcRepositoryApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP
```

Harvester machines

On each harvester machine, we have one or more HarvesterControllerApplications. Each HarvesterControllerApplication have their own sidekick application that restarts the HarvesterControllerApplication after each harvest. Settings related to the HarvesterControllerApplication are

- setting.common.http.port (to distinguish between HarvesterControllerApplications running on same machine)
- settings.harvester.harvesting.queuePriority (to select which of two queues to accept jobs from: HIGH_PRIORITY (jobs part of a selective harvest), or LOW_PRIORITY (jobs part of a snapshotharvest))

In the following, a low-priority HarvestControllerApplication is started with ID=8081

```
 cd $NetarchiveSuiteDir
 export
APP_OPTIONS="-Dsettings.harvester.harvesting.queuePriority=LOW_PRIORITY
\
                    -Dsettings.common.http.port=8081"
 export
APP=dk.netarkivet.harvester.harvesting.HarvestControllerApplication
 java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP_OPTIONS $APP
```

Starting a SideKick application for the just started HarvestControllerApplication

```
 cd $NetarchiveSuiteDir
 export APP_OPTIONS=-Dsettings.common.http.port=8081
 export APP=dk.netarkivet.harvester.sidekick.SideKick
 export
APP_ARGS1=dk.netarkivet.harvester.sidekick.HarvestControllerServerMonitorHook
 export APP_ARGS2=full or relative path to a script that can start the
HarvesterControllerApplication again
 java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP_OPTIONS $APP
$APP_ARGS1 $APP_ARGS2
```

Bitarchive machines

For each Location, you can have one or more machines. We suggest using just one BitarchiveServer for each machine, though it is possible to use more than one. Each BitarchiveServer can have storage on several filesystems, so if archive-storage is spread over more than one filesystem, you need to modify the settings file like this

```
<fileDir>/home/bitarchiveOne/</fileDir>
<fileDir>/home/bitarchiveTwo/</fileDir>
```

Starting a BitarchiveServer requires knowing what Location it resides on, and the credentials required for correcting the data stored in the archive:

```
 cd $NetarchiveSuiteDir
 export
APP_OPTIONS="-Dsettings.archive.bitarchive.thisLocation=ThisLocation \

-Dsettings.archive.bitarchive.thisCredentials=CREDENTIALS"
 export APP=dk.netarkivet.archive.bitarchive.BitarchiveApplication
 java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP_OPTIONS $APP
```

Sccess servers

There are normally access-servers at both Locations. On the access-servers, we deploy any number of **ViewerProxyApplication** instances, and maybe one **IndexServerApplication** (only one in all) used to generate indices needed by the harvesters and the ViewerProxyApplication instances.

```
 cd $NetarchiveSuiteDir
 export APP=dk.netarkivet.archive.indexserver.IndexServerApplication
java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP
```

Each ViewerproxyApplication instance uses a dedicated HTTP port (settings.common.http.port), and its own distinct base directory (settings.viewerproxy.baseDir). They also belong to a Location (settings.archive.bitarchive.thisLocation). In the start sample below, the instance uses HTTP port 8081 and 'viewerproxy_8081' as base directory, and belongs to locationA:

```
 cd $NetarchiveSuiteDir
 export APP_OPTIONS="-Dsettings.common.http.port=8081 \
    -Dsettings.viewerproxy.baseDir=viewerproxy_8081 \
    -Dsettings.archive.bitarchive.thisLocation=locationA"
 export APP=dk.netarkivet.viewerproxy.ViewerProxyApplication
 java $JAVA_OPTS $SETTING $LOG_SETTINGS $JMX_SETTINGS $APP_OPTIONS $APP
```

Starting and stopping the NetarchiveSuite

edit

You need to start and stop the NetarchiveSuite applications in the correct order. The most critical part is that the BitarchiveMonitor must not start before the BitarchiveServers, as it might then try to perform batch jobs without all BitarchiveServers recieving the message. The following is a suggested order of startup:

NetarchiveSuite application startup order

1. The BitarchiveApplication on all bitarchive servers is started:
 `dk.netarkivet.archive.bitarchive.BitarchiveApplication`

2. The applications on the admin-machine are started:

```
 - dk.netarkivet.harvester.webinterface.HarvestDefinitionApplication
 - dk.netarkivet.archive.arcrepository.ArcRepositoryApplication
 - dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication for
Location A
```

```
 - dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication for
Location B
```

3. The applications on the harvester machines are started: For each HarvesterControllerApplication deployed on this machine, the SideKick application is started, which in turn starts a HarvesterControllerApplication

4. The applications on the access-servers are started by first starting the IndexServer and then one or more ViewerproxyApplication instances.

NetarchiveSuite application stopping order

After locating the process-id of any given process, the actually killing of the process is done on unix-machines with the command: `kill $PID`

The killing itself is done in the following order:

1. The applications on the admin-machine are killed:

```
 - dk.netarkivet.harvester.webinterface.HarvestDefinitionApplication
 - dk.netarkivet.archive.arcrepository.ArcRepositoryApplication
 - dk.netarkivet.archive.bitarchive.BitarchiveMonitorApplication
```

2. The BitarchiveApplication on all bitarchive servers are shut down: `dk.netarkivet.archive.bitarchive.BitarchiveApplication`

3. The applications on the harvester machines are shut down in this order: For each HarvesterControllerApplication deployed on this machine, the SideKick application is shut down first, and then the HarvesterControllerApplication. This prevents the SideKick application from starting a new HarvestControllerApplication.

4. The applications on the access-servers are shutdown by first killing the IndexServer and then the ViewerproxyApplication instances.

Remember to empty the JMS queues after shutting down the NetarchiveSuite if you are upgrading the system. If any outstanding JMS messages are around next time the NetarchiveSuite is started, they may cause deserialization errors if the message definitions have changed. To do this, you must recall what JMS environmentName your NetarchiveSuite instance have been using. The details of this are explained in Appendix A below. In the Danish installation, we by default empty the queues any time the system is restarted, so the effect of leaving messages in the queues over a restart even when not upgrading has not been tested in practice.

Monitoring an running instance of NetarchiveSuite

edit

The Status component of the NetarchiveSuite GUI implemented using JMX enables easy monitoring of all running applications. This also detects non-responding machines and applications. All important log messages (Log level INFO and above) can be studied in the

GUI. However, only the last 100 messages from each application are available. This number can be increased or decreased using the setting *settings.monitor.logging.historySize*.

If you want to get more information about the current status of a particular application, you can use the program *jconsole*. You need to know on which machine the the application is running (HOSTNAME), the JMX port (JMX_PORT) and RMI port (RMI_PORT) assigned to the application, and password for the *monitorRole*. Then you just write jconsole, and click on the 'advanced' tab, enter the URL

```
service:jmx:rmi://HOSTNAME:RMI_PORT/jndi/rmi://HOSTNAME:JMX_PORT/jmxrmi
```

When asked for username, enter *monitorRole* and the password set for the application. Log machines can now examined for the given application by seleting MBeans, and unfolding "dk.netarkivet.common.logging".

Appendices

edit

Appendix A : Installing external software

Installing and configuring a JMS broker

The software have been tested with the free JMS broker from Sun "Open Message Queue 4.1", and the commercial JMSBroker "Sun MQ 3.6 Enterprise Edition".

Obtaining JMS

Sun's Open Message Queue can be obtained from the following site: https://mq.dev.java.net/downloads.html Go to the section named "Open Message Queue Binaries", and click on the Linux link in the subsection "Latest Open MQ 4.1 Binary Downloads". This will give you a jar-file named "mq4_1-binary-Linux_X86-XXXXXXXX.jar".

Note: We only support installation on the Linux platform here. However, you may want to install your JMS broker on a different platform. Binary versions are available at the site for: Solaris Sparc, Solaris x86, Linux (x86), Windows (x86). If you want to build a binary for another platform, the source can be downloaded from the download-page.

Installing the JMS broker

Select Linux server where you want to install JMS broker, and select an installation directory. Then log on the linux server as root, and do the following:

```
    export INSTALLATIONDIR = /path/to/installationdir
    cd $INSTALLATION_DIR
    unzip mq4_1-binary-Linux_X86-XXXXXXXX.jar
    chmod +x ./mq/bin/imqbrokerd
```

Check that it starts, and that the last message is <br> ="Broker <localhost>:7676 ready"= We are now ready to configure the JMS broker.

Configuring the JMS broker

- Edit the file $INSTALLATION_DIR/mq/etc/imqenv.conf to set IMQ_DEFAULT_JAVAHOME to a JDK1.5.0.
- Changing the number of the listening port number 7676 is done by editing the line `imq.portmapper.port=7676` in the file
  $INSTALLATION_DIR/mq/lib/props/broker/default.properties
- Set max listeners any given queue to 20. You need to make sure, that the following line `imq.autocreate.queue.maxNumActiveConsumers=20` is present and not uncommented in the file `$INSTALLATION_DIR/mq/var/instances/imqbroker/props/config.prope`

Starting and stopping JMS

The broker is started directly in this way:

```
    $INSTALLATION_DIR/mq/bin/imqbrokerd -reset store -tty &
```

The sysadmin would maybe like to start the broker on machine startup by inserting the statement above into the /etc/rc.d/rc.local

The broker is stopped in this way:

```
logon on machine as root
find processid for the broker (ps auxw | grep imqbrokerd)
kill -9 $IMQ_PROCESSID
```

Alternatively press Crtl-c, if the terminal where the broker was started, is still available

Testing that JMS broker is alive:

```
[svc@udvikling kb-dev-adm-001.kb.dk]$ telnet localhost 7676
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
101 imqbroker 4.1
portmapper tcp PORTMAPPER 7676 [sessionid=1729683678303517696]
cluster_discovery tcp CLUSTER_DISCOVERY 46760
jmxrmi rmi JMX 0
[url=service:jmx:rmi://udvikling.kb.dk/stub/rO0ABXNyAC5qYXZheC5tYW5hZ2VtZW50LnJ
admin tcp ADMIN 46763
jms tcp NORMAL 46762
cluster tcp CLUSTER 46764
.
Connection closed by foreign host.
```

To run JMS client applications, include the following jar files in

the classpath :
$INSTALLATION_DIR/mq/lib/jms.jar $INSTALLATION_DIR/mq/lib/imq.jar

Create a passfile named '.imq_passfile' (used when emptying JMS queues):

```
imq.imqcmd.password=REPLACE_WITH_PASSWORD
```

How to empty queues

log on as root to the server, where the JMS broker is installed. The following assumes that the JMS environmentName is PROD, and that JMS password file resides in ~root/.imq_passfile:

```
export JMS_ENV=PROD
export MQ_HOME=/usr/local
# imqcmd using -u admin -passfile ~/.imq_passfile
$MQ_HOME/bin/imqcmd list dst -t q -u admin -passfile ~/.imq_passfile |
grep ^${JMS_ENV}_ | cut -f1 -d\ |xargs -r -n 1 $MQ_HOME/bin/imqcmd
destroy dst -t q -u admin -passfile ~/.imq_passfile -f -n
$MQ_HOME/bin/imqcmd list dst -t t -u admin -passfile ~/.imq_passfile |
grep ^${JMS_ENV}_ | cut -f1 -d\ |xargs -r -n 1 $MQ_HOME/bin/imqcmd
destroy dst -t t -u admin -passfile
~/.imq_passfile -f -n"
```

Installing and configuring FTP

If you decide to use FTPRemote for file transfer in the NetarchiveSuite, you need to install and start one or more FTP servers, before you begin the installation of the NetarchiveSuite. Any brand of FTP-servers will probably, but we have good experience with Proftpd.

You can download Proftpd from http://www.proftpd.org/. We are using version 1.2.10, but any recent non-beta version will probably do.

The text below the proftpd.conf needed by NetarchiveSuite:

```
# Port 21 is the standard FTP port.
Port                            21

# Umask 022 is a good standard umask to prevent new dirs and files
# from being group and world writable.
Umask                           022

# To prevent DoS attacks, set the maximum number of child processes
# to 30.  If you need to allow more than 30 concurrent connections
# at once, simply increase this value.  Note that this ONLY works
# in standalone mode, in inetd mode you should use an inetd server
# that allows you to limit maximum number of processes per service
# (such as xinetd).
MaxInstances                    30
```

```
# Set the user and group under which the server will run.
User                            nobody
#Group                          nogroup
Group                           nobody

# To cause every FTP user to be "jailed" (chrooted) into their home
# directory, uncomment this line.
#DefaultRoot ~

# Normally, we want files to be overwriteable.
## This is necessary to allow the append operation
AllowOverwrite          on

AllowStoreRestart on

# Bar use of SITE CHMOD by default
<Limit SITE_CHMOD>
  DenyAll
</Limit>

# This enables or disables the PAM authentication module.
# The default is 'on'.
#AuthPAM off
```

Starting and stopping a Proftpd server

Log as root on to the server, where Proftpd is installed, and the following command will start the FTP-server

```
/usr/local/sbin/proftpd
```

The following will kill the FTP-server.

```
killall -9 proftpd
```

Appendix B : Configurable settings in the NetarchiveSuite

The NetarchiveSuite uses a XML file to define settings for its applications. The XML file must conform to the XML schema lib/data-definitions/settings.xsd. The XML file has a specific element for each of the dk.netarkivet.* packages: common, harvester, archive, viewerproxy, and monitor, and then deploy, which also belongs to the monitor packages

In the common part of the settings.xml, we have general purpose settings (settings.common.tmpDir, settings.common.http.port), and settings, that allow us select plugins and their associated arguments(settings.common.RemoteFile.class, settings.common.jms.broker, settings.common.arcrepositoryClient, and settings.common.indexClient.class).

In the harvester part of the settings.xml, we have settings configuring the harvesting process: scheduling, job splitting Most of these settings are used by the HarvestDefinitionApplication

In the archive part of the settings file, we have settings related to archive-access (e.g. certain timeouts, locations and their credentials is defined here). Also behaviour of the BitarchiveApplications is set here.

<!-- In the viewerproxy-part In the monitor-part (used by ????) In the deploy part (only used by the HarvestDefinitionApplication) -->

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://www.netarkivet.dk/schemas/settings"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <common>
        <!-- Common temporary directory for all applications. -->
        <tempDir>./tests/commontempdir</tempDir>
        <!-- FTP connection data-->
        <remoteFile xsi:type="ftpremotefile">
            <!-- The class to use for RemoteFile objects. -->

<class>dk.netarkivet.common.distribute.FTPRemoteFile</class>
            <!-- The default FTP-server used -->
            <serverName>localhost</serverName>
            <!-- The default FTP-server port used -->
            <serverPort>21</serverPort>
            <!-- The default FTP username -->
            <userName>exampleusername</userName>
            <!-- The default FTP password -->
            <userPassword>examplepassword</userPassword>
            <!-- The number of times FTPRemoteFile should try before
giving up
                 a copyTo operation. We augment FTP with checksum
checks. -->
            <retries>3</retries>
        </remoteFile>
        <!-- Connection data for JMS-->
        <jms>
            <!-- Selects the broker vendor to be used. Fx. ActiveMQ.
-->
            <class>SunMQ</class>
            <!-- The JMS broker host contacted by the JMS connection
-->
            <broker>localhost</broker>
            <!-- The port the JMS connection should use -->
            <port>7676</port>
            <!-- The name of the environment in which this code is
running, e.g.
                 PROD, RELEASETEST, NHC,... Common prefix to all JMS
channels
                  -->
            <environmentName>DEV</environmentName>
        </jms>
        <http>
            <!-- The *unique* (per host) port number that may or may
not be
                 used to serve http, but is frequently used to identify
                 the process.-->
            <port>8076</port>
        </http>
        <arcrepositoryClient xsi:type="jmsarcrepositoryclient">
            <!-- The class that implements the ArcRepositoryClient.
This class
                 will be instantiated by the ArcRepositoryClientFactory
-->

<class>dk.netarkivet.archive.arcrepository.distribute.JMSArcRepositoryClient</c
            <!-- How many milliseconds we will wait before giving up on
a
                 lookup request to the Arcrepository. Set to 1 minute
```

```
to
                 make it possible to retrieve large records using FTP
-->
            <getTimeout>60000</getTimeout>
            <!-- Number of times to try sending a store message before
failing,
                 including the first attempt -->
            <storeRetries>3</storeRetries>
            <!-- Timeout in milliseconds before retrying when calling
                 ArcRepositoryClient.store() -->
            <storeTimeout>3600000</storeTimeout>
        </arcrepositoryClient>
        <indexClient xsi:type="indexrequestclient">
            <!-- The class instantiated to give access to indices.
Will be
                 created by IndexClientFactory -->

<class>dk.netarkivet.archive.indexserver.distribute.IndexRequestClient</class>
            <!-- The amount of time, in milliseconds, we should wait
for replies
                 when issuing a call to generate an index over som
jobs.
             -->
            <indexRequestTimeout>43200000</indexRequestTimeout>
        </indexClient>
        <!-- The name of the directory where cache data global to the
entire
             machine can be stored.  Various kinds of caches should be
stored in
             subdirectories of this -->
        <cacheDir>cache</cacheDir>
        <!-- Error notification settings -->
        <notifications>
            <!-- Which class to instantiate to handle error
notifications -->

<class>dk.netarkivet.common.utils.EMailNotifications</class>
            <!-- The receiver of emails -->
            <receiver>example@netarkivet.dk</receiver>
            <!-- The stated sender of emails  (and receiver of
bounces)-->
            <sender>example@netarkivet.dk</sender>
        </notifications>
        <!-- Settings for sending email. Currently mail is only used
for email
         notifications. -->
        <mail>
            <!-- The email server to use -->
            <server>examplesmtpserver.netarkivet.dk</server>
        </mail>
        <!-- JMX logging settings -->
        <jmx>
            <!-- The port to connect to using JMX -->
            <port>8100</port>
            <!-- The RMI port used for communicating with beans -->
            <rmiPort>8200</rmiPort>
            <!-- The password file, containing information about who
may
            connect -->
            <passwordFile>conf/jmxremote.password</passwordFile>
        </jmx>
        <!-- Settings for the web GUI -->
        <webinterface>
            <!-- Language settings -->
            <language>
                <!-- A locale the GUI is available as -->
                <locale>da</locale>
                <!-- Native name of the language for this locale -->
                <name>Dansk</name>
            </language>
            <!-- Language settings -->
            <language>
                <!-- A locale the GUI is available as -->
                <locale>en</locale>
                <!-- Native name of the language for this locale -->
                <name>English</name>
            </language>
            <siteSection>
                <!-- A subclass of SiteSection that defines this part
of the
                     web interface. -->

<class>dk.netarkivet.harvester.webinterface.DefinitionsSiteSection</class>
                <!-- The directory or war-file containing the web
application
                     for this site section.-->

<webapplication>webpages/HarvestDefinition</webapplication>
                <!-- The URL path for this section of the web
interface. -->
                <deployPath>/HarvestDefinition</deployPath>
            </siteSection>
            <siteSection>
                <!-- A subclass of SiteSection that defines this part
of the
                     web interface. -->

<class>dk.netarkivet.harvester.webinterface.HistorySiteSection</class>
                <!-- The directory or war-file containing the web
application
                     for this site section.-->
                <webapplication>webpages/History</webapplication>
                <!-- The URL path for this section of the web
interface. -->
                <deployPath>/History</deployPath>
            </siteSection>
            <siteSection>
                <!-- A subclass of SiteSection that defines this part
of the
                     web interface. -->

<class>dk.netarkivet.archive.webinterface.BitPreservationSiteSection</class>
                <!-- The directory or war-file containing the web
application
                     for this site section.-->

<webapplication>webpages/BitPreservation</webapplication>
                <!-- The URL path for this section of the web
interface. -->
                <deployPath>/BitPreservation</deployPath>
            </siteSection>
            <siteSection>
                <!-- A subclass of SiteSection that defines this part
of the
                     web interface. -->

<class>dk.netarkivet.viewerproxy.webinterface.QASiteSection</class>
                <!-- The directory or war-file containing the web
application
                     for this site section.-->
                <webapplication>webpages/QA</webapplication>
                <!-- The URL path for this section of the web
interface. -->
                <deployPath>/QA</deployPath>
            </siteSection>
            <siteSection>
                <!-- A subclass of SiteSection that defines this part
of the
```

```
                      web interface. -->

<class>dk.netarkivet.monitor.webinterface.StatusSiteSection</class>
                <!-- The directory or war-file containing the web
application
                     for this site section.-->
                <webapplication>webpages/Status</webapplication>
                <!-- The URL path for this section of the web
interface. -->
                <deployPath>/Status</deployPath>
            </siteSection>
        </webinterface>
    </common>
    <harvester>
        <datamodel>
            <domain>
                <!-- Default seed list to use when new domains are
created -->
                <defaultSeedlist>defaultseeds</defaultSeedlist>
                <!-- The name of a configuration that is created by
default and
                     which is initially used for snapshot harvests-->
                <defaultConfig>defaultconfig</defaultConfig>
                <!-- Name of order xml template used for domains if
nothing
                 else is specified (e.g. newly created configrations
use this) -->
                <defaultOrderxml>default_orderxml</defaultOrderxml>
                <!-- Default download rate for domain configuration.
                 Not currently enforced. -->
                <defaultMaxrate>100</defaultMaxrate>
                <!-- This setting describes a regular expression used
to
                     validate domains against. -->

<validDomainRegex>^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+|[^\0000-,.-/:-@\[-`{-\0177]+
            </domain>
            <database xsi:type="derbydatabase">
                <!-- The full URL for connecting to the database.
                     If present and not empty, this URL must match the
settings
                     baseDir and class.-->
                <url>jdbc:derby:harvestdefinitionbasedir/fullhddb</url>
                <!-- The class that handles DB-specific methods -->

<specificsclass>dk.netarkivet.harvester.datamodel.DerbyEmbeddedSpecifics</speci
                <!-- The earliest time of day backup will be initiated,
0..24
                     hours.  At a time shortly after this, a consistent
backup
                     copy of the database will be created -->
                <backupInitHour>3</backupInitHour>
            </database>
        </datamodel>
        <scheduler>
            <!-- Used when calculating expected size of a harvest of
some
                 configuration during job-creation process. This
defines how
                 great a possible factor we will permit a harvest to be
larger
                 then the expectation, when basing the expectation on a
previous
                 completed job. -->
            <errorFactorPrevResult>10</errorFactorPrevResult>
            <!-- Used when calculating expected size of a harvest of
some
                 configuration during job-creation process.  This
defines how
```

```
                 great a possible factor we will permit a harvest to be
larger
                 then the expectation, when basing the expectation on
previous
                 uncompleted harvests or no harvest data at all. -->
            <errorFactorBestGuess>20</errorFactorBestGuess>
            <!-- How many bytes the average object is expected to be on
domains
                 where we don't know any better.  This number should
grow over
                 time, as of end of 2005 empirical data shows 38000 -->

<expectedAverageBytesPerObject>38000</expectedAverageBytesPerObject>
            <!-- Initial guess of #objects in an unknown domain -->
            <maxDomainSize>5000</maxDomainSize>
            <jobs><!-- One Job corresponds to a Heritrix run -->
                <!-- The maximum allowed relative difference in
expected number
                     of objects retrieved in a single job definition.
Set to
                     MAX_LONG for no splitting -->

<maxRelativeSizeDifference>100</maxRelativeSizeDifference>
                <!-- Size differences for jobs below this threshold are
ignored,
                     regardless of the limits for the relative size
difference.
                     Set to MAX_LONG for no splitting. -->

<minAbsoluteSizeDifference>2000</minAbsoluteSizeDifference>
                <!-- When this limit is exceeded no more configurations
may be
                     added to a job. Set to MAX_LONG for no splitting.
-->
                <maxTotalSize>2000000</maxTotalSize>
            </jobs>
            <!-- How many domain configurations we will process in one
go before
                 making jobs out of them.  This amount of domains will
be stored
                 in memory at the same time.  Set to MAX_LONG for no
job
                 splitting. -->
            <configChunkSize>10000</configChunkSize>
        </scheduler>
        <harvesting>
            <!-- Each job gets a subdir of this dir. Job data is
written and
                 Heritrix writes to that subdir-->
            <serverDir>server</serverDir>
            <!-- The directory in which data from old jobs is kept
after
                 uploading.  Each directory from serverDir will be
moved to
                 here if any data remains, either due to failed uploads
or
                 because it wasn't attempted uploaded. -->
            <oldjobsDir>oldjobs</oldjobsDir>
            <!-- Pool to take jobs from -->
            <queuePriority>HIGHPRIORITY</queuePriority>
            <!-- When to stop Heritrix, timeouts in ms. -->
            <heritrix>
                <!-- The timeout setting for aborting a crawl based on
                    crawler-inactivity. If the crawler is inactive for
this
                    amount of milliseconds the crawl will be aborted.
                    The inactivity is measured on the
                    crawlController.activeToeCount(). -->
                <inactivityTimeout>1800</inactivityTimeout>
                <!-- The timeout value (in seconds) used in
HeritrixLauncher
                     for aborting crawl when no bytes are being
received from
                     web servers. -->
                <noresponseTimeout>1800</noresponseTimeout>
            </heritrix>
            <!-- The file used to signal that the harvest controller is
running.
                 Sidekick starts HarvestController if this file is not
present
                 -->
            <isrunningFile>./hcsRunning.tmp</isrunningFile>
        </harvesting>
    </harvester>
    <archive>
        <arcrepository>
            <!-- Absolute/relative path to where the "central list of
files and
                 checksums" (admin.data) is written. Used by
ArcRepository and
                 BitPreservation. -->
            <baseDir>.</baseDir>
            <!-- The names of all institutional bit archive locations
in the
                 environment, e.g., "LocationA" and "LocationB". -->
            <location>
                <name>LocationA</name>
            </location>
            <location>
                <name>LocationB</name>
            </location>
            <!-- Default bit archive to use for batch jobs (if none is
specified) -->
            <batchLocation>LocationA</batchLocation>
        </arcrepository>
        <bitarchive>
            <!-- The minimum amount of bytes left *in any dir* that we
will
                 allow a bitarchive machine to accept uploads with.
When no
                 dir has more space than this, the bitarchive machine
stops
                 listening for uploads.  This values should at the very
least
                 be greater than the largest ARC file you expect to
receive.
            -->
            <minSpaceLeft>200000000</minSpaceLeft>
            <!-- These are the directories where ARC files are stored
                 (in a subdir). If more than one is given, they are
used from
                 one end. -->
            <fileDir>m:\bitarchive</fileDir>
            <fileDir>n:\bitarchive</fileDir>
            <fileDir>o:\bitarchive</fileDir>
            <fileDir>p:\bitarchive</fileDir>
            <!-- The frequency in milliseconds of heartbeats that are
sent by
                 each BitarchiveServer to the BitarchiveMonitor. -->
            <heartbeatFrequency>1000</heartbeatFrequency>
            <!-- If we haven't heard from a bit archive within this
many
                 milliseconds, we don't excpect it to be online and
won't wait
                 for them to reply on a batch job.  This number should
be
                 significantly greater than heartbeatFrequency to
account for
```

```
                 temporary network congestion. -->
            <acceptableHeartbeatDelay>60000</acceptableHeartbeatDelay>
            <!-- The BitarchiveMonitorServer will listen for
BatchEndedMessages
                 for this many milliseconds before it decides that a
batch job
                 is taking too long and returns just the replies it has
                 received at that point. -->
            <batchMessageTimeout>1209600000</batchMessageTimeout>
            <!-- For archiving applications, which bit archive are you
part of?-->
            <thisLocation>SB</thisLocation>
            <!-- Credentials to enter in the GUI for "deleting" ARC
files in
                 this bit archive -->
            <thisCredentials>examplecredentials</thisCredentials>
            <!-- When the length record exceeds this number, the
contents of the record
                 will be transferred using a RemoteFile. Currently set
to 10 MB
              -->

<limitForRecordDatatransferInFile>10485760</limitForRecordDatatransferInFile>
        </bitarchive>
        <bitpreservation>
            <!-- Absolute or relative path to dir containing results of
                 file-list-batch-jobs and checksumming batch jobs
                 for bit preservation-->
            <baseDir>bitpreservation</baseDir>
        </bitpreservation>
    </archive>
    <viewerproxy>
        <!-- The name of the server used for viewerproxy links in the
GUI. -->
        <hostName>kb-dev-acs-001.kb.dk</hostName>
        <!-- The main directory for the ViewerProxy, used for storing
the Lucene
             index for the jobs being viewed -->
        <baseDir>viewerproxy</baseDir>
    </viewerproxy>
    <monitor>
        <!-- The name of the application, fx.
"BitarchiveServerApplication".
             The monitor puts this with each log message -->
        <applicationName>NA</applicationName>
        <logging>
            <historySize>100</historySize>
        </logging>
        <!-- Log dir for the unused Monitor application, wherein logs
used
for creating monitoring pages in the GUI are dropped.-->
        <htmlLogsDir>./log/</htmlLogsDir>
    </monitor>
    <deploy>
<jmxMonitorRolePassword>JMX_MONITOR_ROLE_PASSWORD_PLACEHOLDER</jmxMonitorRolePa
<numberOfHosts>1</numberOfHosts>
<host1>
<name>localhost</name>
<jmxport>8100</jmxport>
</host1>
</deploy>
</settings>
```

Appendix C : Plugins in the

All the settings above ending on ".class" indicate that there is a choice of several classes to choose from. It is not always correct. But our framework does at least enable the installer to replace the default class with a class of his own, or an already existing class (if available).

We now describe the available plugs, and existing plugins for these plugs.

settings.common.remoteFile.class: This setting allows you to select your chosen way of filetransfer in the NetarchiveSuite. You can here choose between FTPRemoteFile (where the data is transferred using a FTP-server), and HTTPRemoteFile (where the data is transferred using a two embedded webservers (one at each end). [Firewall-note] The HTTPRemoteFile requires dedicated ports to be open between all possible senders and recipients of data. For implementers of new filetransfer methods, this class must implement the class dk.netarkivet.common.distribute.RemoteFile. The default value is FTPRemoteFile.

Select a datatransfer method (RemoteFile) [Move to InstallationManual - preinstallation_configuration]

settings.harvester.datamodel.database.specificsclass: This setting allows you select which type of database you want to use. There are support for 3 types already: An Embedded Derby database (dk.netarkivet.harvester.datamodel.DerbyEmbeddedSpecifics), an external Derby database (dk.netarkivet.harvester.datamodel.DerbyClientSpecifics), or an MySQL database (dk.netarkivet.harvester.datamodel.MySQLSpecifics). The default is DerbyEmbeddedSpecifics. If you choose not to use the default, you need to replace the default database URL(setting settings.harvester.datamodel.database.url), and maybe the time for the daily backup to start (setting settings.harvester.datamodel.database.backupInitHour))

settings.common.jms.broker SunMQ (designates, that the ....). must implement the JMSConnection class.

settings.common.arcrepositoryClient.class. Must implement dk.netarkivet.common.distribute.ArcRepositoryClient The available choices are the default dk.netarkivet.archive.arcrepository.distribute.JMSArcRepositoryClient (that is required, if you want to access the distributed type of archive that is included in the NetarchiveSuite). and the dk.netarkivet.common.distribute.TrivialArcRepositoryClient (allows for access to a local archive)

settings.common.notifications.class: Allows for different ways of making notifications. The default and only choice is the class dk.netarkivet.common.utils.EMailNotifications (which allows you receive notifications by email). The use of this plugin requires setting the mail-server, the recipient- and sending email-address

setting.common.webinterface.sitesection.class This setting allows you to add webmodules to the NetarchiveSuite GUI. More than one of these SiteSection classes can be active in the same GUI. the default(standard) configuration contains all 5 existing webmodules:

1) HarvestDefinition: Allows you to define and schedule harvests , 2) HarvestHistory: See the status of running and finished harvestjobs 3) BitPreservation: This module allows us to ... data in our archive 4) QA: Module for doing Quality Assurance 5) Status: Module for monitoring the health of all machines and applications

setting.common.webinterface.language: The languages supported by the webinterface. Danish (locale=da)and English (locale=en) are supported currently. The DeveloperManual will tell you how to add support for more languages to the NetarchiveSuite.

*settings.common.indexClient:

```
<indexClient xsi:type="indexrequestclient">
            <!-- The class instantiated to give access to indices.
Will be
                 created by IndexClientFactory -->

<class>dk.netarkivet.archive.indexserver.distribute.IndexRequestClient</class>
            <!-- The amount of time, in milliseconds, we should wait
for replies
                 when issuing a call to generate an index over som
jobs.
             -->
            <indexRequestTimeout>43200000</indexRequestTimeout>
        </indexClient>
```